Interactive out-of-core visualization of very large multiresolution time series scientific data
Progress Report
April 15, 2007

1. Proposed goals for year 1

The principal goals for the year as described in the proposal are shown in the table below.

Development Task Target	Month	Scientist Task Target
1. Software framework	3	Access our data modules from their renderer code
2. Granite/C++	5	Use caching/pre-fetching features, compare results
3. Visualization application	7	Use and evaluate our visualization application
4. LAN distributed data	9	Distribute their active data over the LAN, evaluate
5. Non-uniformly distributed grid	12	Modify renderers to access grid position data

The first two development goals (*Software framework* and *Granite/C++*) address critical foundational tools to provide the key software support for efficient access to the multiresolution (MR) data sets generated by our MR data generation tool.

The third goal (*Visualization application*) was the key factor that drove the entire implementation strategy; we planned to develop visualization algorithms that use our multiresolution data access software. Our goal was to implement specific visualization algorithms that would be useful to support the research of our science collaborators doing MHD research.

The fourth goal (*LAN distributed data*) was intended to evaluate the performance characteristics associated with distributing the multiresolution versions of our colleagues' very large data sets over the local area network. In particular, we want the lower resolution data representations to be stored on the local workstation and higher resolution data stored in a large file server on the LAN. We intended to evaluate the performance degradation that results from the higher latency of LAN access versus local disk access in order to develop guidelines for a heuristic determination of appropriate distribution parameters.

The final goal (*Non-uniformly distributed grid*) was to enhance our MR data software to support non-uniform grid positioning.

2. Revised goals for year 1

In July 2006, we developed a significantly different implementation model than we had proposed. Our science collaborators had began using the *VisIt* visualization environment developed at Lawrence Livermore Laboratories. This package provides a large number of visualization modules in a user-friendly interactive environment that also supports parallel computation on their very large Beowulf cluster. The *VisIt* environment was a very effective tool for their research needs. In consultation with our collaborators we decided to re-evaluate our goals; it was not feasible for us replicate all of the visualization options that *VisIt* already provides. It became clear that rather than implementing specific visualization algorithms using our MR data management tools, we should develop a *VisIt* data interface that would allow all *VisIt* visualization modules to access our MR data management software. The principal effect of this decision was the replacement of our third major goal (*Visualization application*) with one to develop a *VisIt* data interface module and a re-ordering of the implementation schedule. This revised plan is shown in the table below.

Revised Development Goals	Revised Scientist Goals
1. Software framework	Access our data modules from their renderer code
2. Non-uniformly distributed grid	Modify renderers to access grid position data
3. *VisIt* data interface module	Use and evaluate the *VisIt* interface
4. Granite/C++	Use caching/prefetching features, compare results
5. LAN distributed data	Distribute their active data over the LAN, evaluate

3. Achievements for year 1

3.1. Software framework (STARview)

As a result of our decision to put more effort into integration with existing visualization tools (in particular, *VisIt*), we altered our plans for development of our own visualization environment. Instead of concentrating on *visualization tool* development, we re-examined and re-designed our software framework to better isolate the *data modeling* components from the visualization components. This was an aspect of our initial design, but the realities of interfacing with a completely independent software package exposed some weaknesses in our original design. We had also gained experience with using our initial design that identified other modifications that we felt would create a cleaner, more efficient design. We ended up doing a fairly extensive re-design and re-implementation of the *Datasouce* subsystem of *STARview*. The resulting system is a significant improvement in elegance, flexibility, maintainability, and performance. The revised *Datasource* module provides a simple straightforward interface to the *VisIt* data interface module.

3.2. Non-uniformly distributed grid

We implemented support for the kinds of non-uniformly distributed grids that are needed by our collaborators. In this case grid positions are rectilinear, but the positioning along each axis is variable. Such data is often called a non-uniform rectilinear grid. The positioning information is specified in a metadata file whose presence automatically triggers the invocation of this support.

3.3. VisIt data interface module

VisIt is a general purpose visualization environment aimed at giving researchers scientific visualization tools for scientific datasets. *VisIt* is built largely upon the *Visualization Toolkit (vtk)* libraries, extending the interface and providing a comprehensive environment to the scientist. The *VisIt* toolkit supports many different types of data and has a modular architecture that allows users to build data plugins to access other types of data. We have implemented a database plugin for *VisIt* that can read our multiresolution hierarchy and provide *VisIt* renderers with multiple resolutions of data.

We have integrated our multiresolution data software into *VisIt* so that any rendering plugin can use the multiresolution data. Existing plot plugins in *VisIt* do not even need to be recompiled to become *multiresoluion-aware*. To achieve this task, we have separated our multiresolution data access module from the visualization subsystem.

Our software consists of two *VisIt* plugins. The first is a *Database Reader* plugin, which has the responsibility of reading and importing data in our multiresolution format for use with *VisIt* plots. The second plugin is an *Operator* plugin, which presents the user with a widget, implemented as an *Operator Attribute*, to control the level of refinement from which the current plot plugin gets its data. By checking the box *Auto Update* in the *VisIt* user interface, changes to the current resolution will automatically re-invoke the *VisIt* rendering pipeline, so that the current plot is regenerated with the chosen level of refinement.

Technically speaking, our multiresolution control widget does not "operate" on the data in the sense defined by the *VisIt* framework (i.e., as a run-time filter). Instead, this widget acts only as a front end controller to the database plugin, directing it to fetch data at user specified resolutions. A more proper implementation would not use an *Operator* for such a task. Instead, a generic controller widget associated only with the *Database* plugin should be implemented. We are told that future versions of *VisIt* will support such a feature, and it is our intention to use that functionality when it becomes available.

There is a fundamental and significant limitation to the multiresolution functionality that we can support via the *VisIt* interface. *VisIt* relies on the *vtk* for its underlying visualization model and tools. The *vtk* implementation model is very heavily dependent on the assumption that all data to be rendered can be stored in main memory at one time during the rendering processes. It is simply not feasible to retrofit existing *vtk-based* renderers to perform *out-of-core* rendering, which is what is needed to create an image from data sets that are larger than will fit in main memory. It is possible, of course, to write a specific *new* renderer that does *out-of-core* rendering, but that approach does not achieve our goal of keeping the renderer ignorant of the implementation of the data access.

Our *VisIt* interface is a very effective mechanism for accessing multiresolution data sets without having to rewrite each visualization tool – as long as the data to be rendered fits in main memory. This is not a huge restriction given the basic interaction model whereby higher resolution data representations are accessed as the user zooms in to smaller spatial and temporal ranges – the system can use available memory as a constraint in determining which resolution to access for a given spatio-temporal request. However, there are significant features of our proposed environment that are not effective in this context. In particular, we have shown dramatic improvements in interactive visualization performance using our *iteration-aware* caching and pre-fetching techniques for *out-of-core* rendering. We cannot provide this functionality within the current *VisIt* environment.

3.4. Granite/C++ interface

The *Granite* component of our software environment is a comprehensive Java-based environment for modeling and interfacing to very large scientific data sets. This software was developed as part of two previous NSF grants for research in this area. Of particular relevance to our AISR efforts are the features *Granite* provides for doing efficient I/O caching and pre-fetching that provides dramatically improved performance for *out-of-core* data visualization compared to what is possible using the normal file-system caching. An important goal of this project is to make these features of *Granite* available to *STARview* by developing a *Granite/C++* interface.

Because of the nature of the current *VisIt* environment, the *VisIt* rendering modules cannot take advantage of the *Granite* support for *out-of-core* visualization. Consequently, when we re-defined our goals in July, the *Granite/C++* interface lost some of its immediate priority. We still think this an important functionality that will become increasingly valuable as data sets continue to grow in size. The immediate value of providing multiresolution support for <u>all</u> *VisIt* renderers, however, was clearly much higher.

We have, however, made progress in implementing this interface. The basic data access functionality is working; the *STARview* C++ environment can open and access any datasource via a *Granite RemoteServer* process. The *RemoteServer* also supports *iterator*-defined access, which is the key feature needed to access the *Granite* caching and pre-fetching functionality. However, it does not yet transform the iteration-specification into the appropriate *Granite* calls to invoke the caching and pre-fetching code. Performance tests of the implemented functionality show a small, but significant degradation in access time when using the *Granite* interface compared with direct C++ I/O. The overhead, however, is insignificant compared to the gains we expect to achieve in *out-of-core* visualization when the caching and pre-fetching functionality is added.

3.5. LAN distributed data

We have not yet carried out a rigorous performance test for access to distributed data over a LAN. Our informal experience indicates that there is a noticeable difference in performance between accessing data on a local disk compared to data on a LAN. This degradation is partially mitigated within the *Granite* system by the caching and pre-fetching functionality. We would prefer to carry out real testing after that functionality is implemented in the *Granite/C++* interface. In addition, it is pretty clear that the need for accessing LAN data is crucial and that the performance degradation is acceptable. Although we would like to try to quantify that better and try to develop guidelines for the nature of the distribution, this does not seem to be nearly as important as other possible tasks. Finally, we have a colleague, Philip Rhodes, at the University of Mississippi who has been extending *Granite* to support caching and pre-fetching functionality over a wide-area network (WAN). His preliminary results have been promising and the techniques he is developing could be also be applied to reduce latency in data access over a LAN. Again, it would be preferable to do careful LAN performance testing that could incorporate this option as well.

3.6. Achievements summary

The table below provides a very terse summary of what we have accomplished for each of our *revised* major goals.

Revised Goal	**Achievements**
1. Software framework	Planned implementation was completed; additional re-design and re-implementation completed in order to better support *VisIt* integration.
2. Non-uniformly distributed grid	Completed for non-uniform rectilinear data distribution.
3. *VisIt* data interface module	Preliminary version completed and is being used by our collaborators. This version requires the user to explicitly invoke a resolution level change.
4. Granite/C++	Basic interface working; but caching and pre-fetching functionality is not yet complete.
5. LAN distributed data	Not done. Now seems to have lower priority

4. Revised future goals

4.1. Investigate VAPOR

VAPOR is a 3D flow visualization environment being developed at NCAR. *VAPOR* is much more focused than *VisIt*, but it seems to provide some very sophisticated flow visualization tools for which there is nothing comparable in *VisIt. VAPOR* also incorporates on-the-fly MR computations based on wavelets. The *VAPOR* approach to MR data representation is significantly different from ours, but the overall software structure of their system may allow us to integrate our data model into their code. We have downloaded and installed *VAPOR*. We intend to investigate whether it is feasible and desirable to try to do this integration. The input from our science collaborators concerning the appropriateness and value of the interactive visualizations provided by *VAPOR* will be a critical factor in determining the desirability of this task. (Note that the complete task list only includes *VAPOR* evaluation. If we decided that an interface to our data model is desirable, that will become a new task that will probably have to displace some other task.) Target completion: July 2007.

4.2. Non-uniform grids

Our current implementation only provides limited support for non-uniformly placed grid positions; we support the non-uniform rectilinear grid used by our collaborators. A more general support would allow a user to identify specific attributes in the data that represent the positions of each point in the grid. Such support would be highly desirable prior to making the software available for general use. We plan to add this feature to the software. Target completion: August 2007.

4.3. Develop distribution package for the VisIt interface and STARgen tools

The current version of the *VisIt* plugin interface to our multiresolution data model is essentially complete and very usable. It will have a broader use once we provide better support for non-uniform grids. We would also like to improve the user interface to make it easier for the user to browse through a data hierarchy tree. Once that is done, we plan to bundle this tool, the associated data generation tool, *STARgen,* and some documentation into a distributable package for anyone to download. We believe that this interface will be of value to a wide range of users. *STARgen* output is used by the *VisIt* interface. *STARgen* input works on any data that is represented as 2D or 3D arrays and is stored with one attribute per file. It also supports time series data as long as all the files of a single attribute are stored in the same directory and the names of the files follow a pattern that includes the time step index as part of the name. Target completion: September 2007.

4.4. VisIt interface enhancement

The current *VisIt* interface tool requires the user to explicitly select the desired resolution level. Our *STARview* tool automatically selects the highest resolution level that "fits" into a specified memory footprint chosen to insure some desired level of interactivity. We would like to bring this functionality to the *VisIt* environment. (Note that we do not intend to provide this functionality in our initial distribution package described above; we do not want to distribute software until we have used it fairly extensively ourselves.) Target completion: September 2007.

4.5. Error support

A fundamental aspect of our entire data model is that we integrate local error estimates into the lower resolution data representations. It is very important that a scientist be able to see the *authenticity* of any visualization being presented from processed data. Integrating error representation into the <u>same</u> visualization as the data itself is a very complicated research problem that we do not plan to address directly. On the other hand, it is not very hard for a user to generate two side-by-side images – one with the data and one with the error – or to alternate a data view with an error view. This is especially easy in the context of the *VisIt* environment. To make this work, however, we need to finalize our error-generation utility that computes an approximation to a local error model for every low resolution data representation. Target completion: September 2007.

4.6. Granite/C++ interface

We plan to expand the *Granite/C++* interface to provide access to the *Granite* caching and pre-fetching functionality. Since it is not likely that this functionality will be beneficial within the *VisIt* environment, we plan to test this from our *STARview* software. The potential benefit of this functionality is significant enough that we need to evaluate its power even if we can only do that by implementing a new visualization tool to access it. Target completion: January 2008.

4.7. Adaptive resolution support

Adaptive resolution (AR) data representation provides a single data representation that has different resolutions in different spatial or temporal regions; regions with high variation will use a higher resolution and than those with low variation. This model may allow for significantly reduced memory utilization and I/O time. Target completion: January 2008.

4.8. VisIt AR interface

Once we have a reasonable AR implementation, we want to provide access to it from the *VisIt* environment. Target completion: July 2008.

4.9. AR caching/pre-fetching support

Caching and pre-fetching strategies are significantly more complicated for AR data representations than they are for uniform resolution data. Target completion: July 2008.

4.10. Grid access

One of our primary long-term goals is to be able to integrate our data management software tools into a Grid environment. Target completion: January 2009.

4.11. Performance studies for system parameter heuristics

A previous target goal was to evaluate the performance characteristics of data access to a local disk compared to data access over a local area net (LAN) with the stated goal of trying to provide user guidance for distributing the data at different resolution levels. This particular performance study is just one of many that could be carried out to provide guidelines to help users configure a variety of system parameters. We certainly plan to do performance tests at every stage of the development in order to validate the effectiveness of our approach. We believe now, however, that tests aimed at "tuning" the system parameters would be most effectively carried out near the end of the grant period. At that time we will have a much better understanding of which parameters are important to tune and which are not. Consequently, we now plan to incorporate such tests as a goal for the last quarter of the grant period. (Note that this goal has replaced the original last goal from the proposal, which was to experiment with the integration of simulation and sampled data into a single visualization. This would have been a completely new research effort initiated very near the end of the grant period. As interesting and important as this task is, it seems like the overall goals of this project and the AISR program would be better served by focusing the end game on this new goal.) Target completion: March 2009.

4.12. Summary of major goals

The table below provides a summary of the major planned tasks for the remainder of the grant period. Note that there will be additional code packaging and distribution tasks for subsequent versions of the software. Specific dates for completion of these tasks will be determined later.

Development Task	*Date*	*Science task*
1. Investigate *VAPOR* as another potential framework environment	7/07	Give us advice/feedback on value of *VAPOR* visualization tools

2. Expand non-uniform grid support	8/07	None. Their data format is already supported.
3. Package *STARgen* and *VisIt* interface for distribution	9/07	Test ease of installation.
4. *VisIt* data interface: auto resolution change	9/07	Use and evaluate modified interface
5. Error support	9/07	Use and evaluate error functionality
6. Granite/C++: caching/prefetching	1/08	Use caching/prefetching features, compare results
7. Adaptive resolution prototype	1/08	Test our implementation tool and give feedback
8. VisIt data interface: AR support	7/08	Compare performance/quality to non-AR using *VisIt*
9. Adaptive resolution cache support	7/08	
10. Grid access	1/09	Use and evaluate data access over the grid
11. Performance studies and system parameter guidelines	3/09	

5. Publications

Foulks, R., D. Benedetto, R.D. Bergeron and T.M. Sparr, STARdata: A Data Server for Multiresolution Time Series Data, *AGU Fall Meeting Abstracts,* Dec. 2006, p. A1316+.

Bergeron, R.D. and R.A. Foulks, "Interactive Out-of-Core Visualization of Multiresolution Time Series Data", in *Numerical Modeling of Space Flows*: *1ST IGPP – CalSpace International Conference*, ed. Nikolai V. Pogorelov and Gary P. Zank, ASP Conference Series, Vol 359, Astronomical Society of the Pacific, 2006, pp. 285-294.